Efficient global computations on a processor network with programmable logic

Jean-Marie Filoque, Eric Gautrin, Bernard Pottier

UNITÉ DE RECHERCHE
INRIA-RENNES

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P. 105
78153 Le Chesnay Cedex
France
Tél.:(1) 39 63 55 11

Rapports de Recherche

N° 1374

Programme 1
Architectures parallèles, Bases de données,
Réseaux et Systèmes

EFFICIENT GLOBAL COMPUTATIONS ON A PROCESSOR NETWORK WITH PROGRAMMABLE LOGIC

Jean-Marie FILOQUE
Eric GAUTRIN
Bernard POTTIER

Janvier 1991

RR-1374

Efficient Global Computations on a Processor Network with Programmable Logic*

Calcul Efficace d'un Etat Global sur un Réseau de Processeurs couplé à de la Logique Programmable

Jean Marie Filloque*, Eric Gautrin**, Bernard Pottier***

*LIBr-ENST Bretagne, Kernevent-Plouzane, 29285 Brest
**IRISA-INRIA, Campus de Beaulieu, 35042 Rennes Cedex
***LIBr-UBO, UFR Sciences, av. Le Gorgeu, 29287 Brest

January 14, 1991

Publication Interne n° 562 - Novembre 1990 - 14 pages

Abstract: A new parallel MIMD architecture is described each node of which is tightly coupled to a global programmable logic layer. This layer gives local acceleration to the node processors by massive micro-grain parallelism. It also provides fast computation services to distributed algorithms by synthesis of global dedicated units operating directly on node operands. As a result, fine approximations of global states become transparently visible in each node, in contrast with usual difficulties and delays in sharing and computing control data.

This point is emphasized by the description of two parallel virtual time mechanisms. The first study involves increasing virtual clocks, and the second one takes into account time counter overflows in a time warp environment. Implementations are based on global systolic networks, fed by the array of local operands and controlled by a small automaton. So, global states are handled for each cycle of the mechanism, and results become visible after one pipeline delay with no cost for the accelerated parallel machine.

Summarized general chararacteristics of this architecture are: general purpose, reconfigurability, cheapness, extensibility.

Résumé: Nous décrivons une nouvelle architecture MIMD parallèle dont chaque nœud est fortement couplé à une couche de logique programmable partagée. La couche programmable permet d'accélérer les calculs d'un noeud en autorisant un parallélisme massif à grain fin. Elle apporte également un service de calcul rapide pour les algorithmes distribués en autorisant la synthése d'unités de calcul partagées et dédiées qui opèrent directement sur les opérandes des nœuds. En conséquence, une approximation fine d'un état global devient disponible de manière transparente sur chaque nœud. Ceci contraste avec les usuelles difficultés et délais rencontrés pour le partage et le calcul de données de contrôle dans les machines parallèles.

Ce dernier point est illustré par deux mécanismes de gestion du temps virtuel. Le premier cas s'intéresse à des horloges virtuelles croissantes, et le deuxième prend en compte les débordements de compteurs de temps dans un environnement optimiste de type "time warp". Les mises en œuvre sont basées sur le principe d'un réseau systolique partagé alimenté par un tableau d'opérandes locaux, et controlé par un petit automate. Grâce à ce mécanisme, un nouvel état global est délivré à chaque cycle, et diffusé à tous les nœuds après un délai de pipeline sans coût pour la machine parallèle accélérée.

En résumé, les caractéristiques générales de cette architecture sont: généralité, reconfigurabilité, faible coût et extensibilité.

*This work is supported by *Région Bretagne* and *Municipalité de Brest*. The Armen machine implementation is supported by *ANVAR*.

Introduction

Flexible architectures have proven to help software developments considerably. Two examples are: (i) reconfigurable processor networks where communication support eases node to node exchanges[10], (ii) writable control stores giving emulation of various instruction sets[8]. Beside the strong interest for fast and flexible interconnection networks, a very attractive hardware support in an MIMD machine could be some global unit receiving data from node interfaces and sending computation results back to these nodes. To cover large fields of applications, it is necessary to define such a unit with a technology allowing deep modifications of its behavior. Recent advances in reconfigurable logic technology have given the opportunity to investigate all kinds of global hardware supports to accelerate control and computation in parallel architectures.

This paper presents the architectural concept of *global reconfigurable coprocessors* for MIMD machines. For this purpose, local *reconfigurable logic sockets* are added to each node and connected together to build a *linear logic layer*. The topology of the parallel machine does not need to be specified, but there is a requirement for some stable primary communication services. To get additional hardware support, the operating system must synthesize services into the logic layer. This task is achieved by sending first configuration specifications to each node, and then writing them into the configuration memory of the local socket. Delays for this last task are currently from 0,1s to 1s, and this process can be repeated and interleaved with execution.

The addition of a reconfigurable logic layer to an MIMD machine has two strong advantages with respect to technology and architecture. First, reconfigurable logic *is* an integration technology and allows very efficient circuits to be synthesized and used. Second, the coprocessor has a strategic position in the MIMD machine. It is strongly tied to each node of the machine, but conserves properties of a dedicated centralized functional unit. It can improve intensive computations as a local accelerator, or distributed computations as a global coprocessor.

The objectives of this paper are twofold:

1. the configurable layer use is illustrated and demonstrated by a description of two Global Virtual Time coprocessors for distributed algorithm support. It is shown by these simple examples that inefficient software tasks can be improved in a smart way by the reconfigurable layer.
2. an original algorithm is proposed where a global controller is synthesized to synchronize the nodes periodically. The period is an application dependent tunable constant.

The paper is organized as follows:

- The first part is a short description of architecture principles. The general coprocessor status is emphasized by the notion of dedicated synthesized architecture.
- In the second part we introduce some notions from the distributed simulation field, and describe an implementation of global virtual time computation on the configurable layer.
  - A minimum approach is first investigated without any attempt to manage the time counter overflow. The synthesized service can be used to prevent mutual drift between the logical local clocks.
  - A second approach is virtual time management processes on user-specified time slice boundaries.

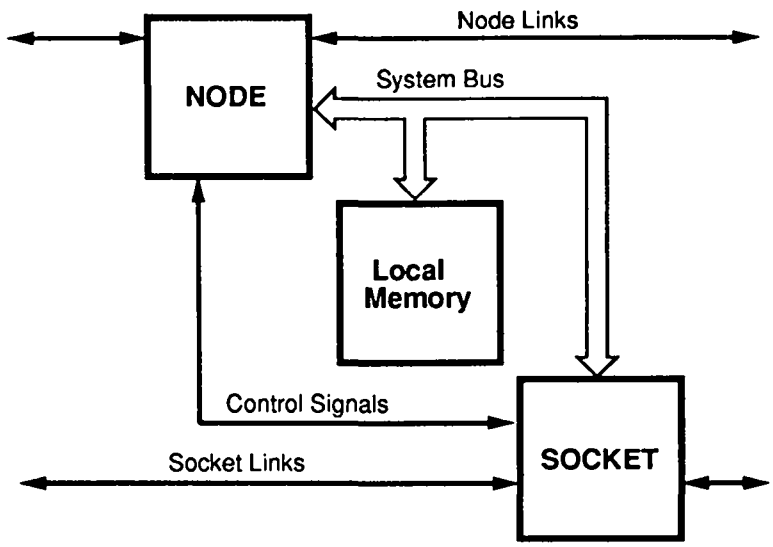

Figure 1: Node Architecture

Information relative to a practical implementation and the whole project is given, and we conclude by general considerations and fields of application.

1 Accelerated parallel architecture

1.1 Node architecture

The proposed architecture principle involves a general purpose parallel machine with a shared or distributed memory, and a complementary global synthesized coprocessor. Figure 1 shows a node with a processor, local memory and a configurable socket. The socket interconnection has a ring topology.

The socket can be implemented with a large commercial reconfigurable logic array providing at least three data ports to the local system bus and the two adjacent sockets. The local interface of the socket is connected to the processor interrupt and arbitration signals, as well as to local memory control lines. Access to the configuration memory of the socket is mapped into the processor address space, and normal memory processor transactions are passed to the socket logic to be internally interpreted. Therefore the socket can be seen as a second processor rather than a slave unit.

1.2 Synthesized coprocessors properties

Coprocessors can operate on control information, instructions and data[14]. They can be synthesized to yield three distinct classes of computation:

- local coprocessing, examples of which are : instruction set emulations, data intensive algorithms, support for heavily used functions.
- massive parallel computations on the reconfigurable layer which may be used as a large operator controlled and fed by the parallel machine. Two fields of application are systolic signal processing and cellular automata.

- architectural support for global control of the parallel machine. Expected applications are load balancing, fast termination detection, global synchronization and virtual clock support.

The two following paragraphs describe the architecture's design and properties of the reconfigurable logic technology respectively.

Sockets are tightly coupled to node processors and embedded into the reconfigurable layer, thus providing local interfaces to global synthesized operators. Data are directly used as operands for calculations, avoiding the very heavy control process of an MIMD machine, where objects must be carried from node to node to be processed on Von Neumann processors. Therefore there is no longer a bottleneck from bus or network contentions, and there are no prohibitive delays from transport layers. Global intensive computations can be achieved on networks of self-synchronized operators, possibly controlled by a small automaton on a special node. This property applies either to highly regular algorithms or to the control support of irregular applications. This last point has a predecessor in the *Fetch-And-Op* primitive operation of the Ultracomputer [3], where a dedicated network provides a global service with implicit mutual exclusion. Another accurate comparison is the systolic architecture, which is known to minimize regular application control by allowing fast communication between interconnected processors. In a similar way, reconfigurable coprocessors provide tight coupling between nodes and global services, thereby greatly improving control and synchronization for irregular distributed applications.

Von Neumann processors use fixed size general operative units, sequence test and execution, reject constants into data or program space. Accessing data involves the use of memory tables or register files. In contrast to these processors, a synthesized operative unit matches the operand size, implements test and execution in parallel and integrates temporarily stable data into the operators. Memory tables can be mapped into internal trees with very fast access time. As a result, synthesized logic efficiently implements massive micro-grain parallelism. Previous work from various authors has taken advantage of these properties to allow considerable speedups for many applications like image processing, encryption, data compression, long integer arithmetics [1, 7, 13, 17]. The proposed architecture will obviously benefit from the technology, enlarging node fields of application.

The following section shows the need for development activities to build synthesized dedicated architectures on very general purpose hardware.

1.3 Coprocessor development model

Considering a conventional working application we can distinguish three logical components:

3	Application software
2	System support
1	Machine hardware

Each layer in the machine brings services and constraints to the upper layers. As a result application software must deal with the characteristics of underlying components. A common alternative to general purpose machines are specialized ones with the following problems: (i) small market segments involve higher costs, (ii) hardware and software investments are more difficult to preserve. The programmable logic layer architecture introduces an additional flexible component into the usual decomposition, allowing temporarily specialized machines to be synthesized on general hardware :

4	Application software
3	System support
2	Programmable logic layer
1	Machine hardware

Such a machine inherits properties of the conventional initial hardware, because of the transparency of *layer 2*. The behavior of a specialization is similar to the addition of optional arithmetic or dedicated[1] coprocessors to existent machines. It becomes possible to obtain efficient dedicated services by rejecting some difficult points of software implementation into programmable logic. Another point of interest is that new applications are more independent of technology: a specific configuration is defined to match the problem exactly, and the influence of processor integration advance is minimized.

Layer 2 definitions come from creations of *configuration files*. This is currently a CAD activity, similar to peripheral driver writing at operating system level, but there is room to design more dynamic schemes. The next section will show two virtual time services for distributed algorithms. It is envisioned that such services can be part of independent resource libraries to be released for application developments. Speed-ups and additional supports are two benefits from the proposed architecture on the quantitative and qualitative sides.

2 A first example of a global virtual time service

Introduction

Distributed systems with pure message passing communication usually use *logical clocks* to timestamp events and messages used to bring them from one process to another. Lamport has shown in [11] that it is possible to construct a total order over their occurrences by using strictly increasing counters, incremented on each emission and updated at reception. So, throughout the system, reception always occurs *after* emission.

In the distributed simulation domain, the problem of virtual clocks (another name for logical clocks) is a little bit more complicated because these clocks are not completely unrelated. Jefferson [5] has proposed the paradigm of *virtual time* that coordinates execution with an imaginary virtual clock. *Virtual time* represents global information and each site can have only an approximation of it. *Virtual time* can be implemented with either a pessimistic or an optimistic approach. With the first, a process on a site can safely increase its local clock only if it is sure that it will receive no message in its past. The respect of this *causality* constraint may lead to deadlock. This approach is presented in [12]. It consists in avoiding or resolving deadlocks. Local virtual clocks, as well as *virtual time* never decrease. The second approach assures only the growth of *virtual time* but not of local clocks. So it allows rollbacks in the past to occur on a site. This is described in [4, 5].

It is, a priori, impossible to have a consistent view of global state and time in such an environment without a shared memory and a common clock. So, processes must content themselves with a best possible approximation of this global information. The construction of a global time approximation is proposed by several authors. It consists in a steady evaluation of a lower bound of all the local clocks in the network. This approximation is used to prevent mutual drift between logical clocks like in [15], to update queues and to avoid memory saturation in time warp systems like in [4, 16], to estimate load ratio of processors for load balancing... This type of computation is suitable for implementation in the programmable layer of the machine, and the following sections describe two applications used to support this assertion.

2.1 A global computation for increasing virtual time

The goal of this section is to emphasize the use of basic mechanisms to build coprocessors. This presentation is driven by the example of a global computation for increasing virtual time. A coprocessor is synthesized to swiftly calculate a lower bound of all *Local Virtual Time* with a circulating token and

[1] An example is the coprocessor board for parallel simulation proposed in [2]

so, to deliver either this bound (a fine approximation of the global virtual time) or an upper limit for message emission timestamps to each node . This limit may also be evaluated in the programmable layer. It is a simple addition with a constant.

For the sake of simplicity, this first proposal does not manage time counter overflows. The coprocessor must compute a *Global Virtual Time* as the minimum of each node *Local Virtual Time*. For the following, let us define GVT to be an evaluated *Global Virtual Time*, and LVT to be a node *Local Virtual Time*.

2.1.1 Coprocessor Architecture

To achieve global evaluation, the coprocessor will receive local data, like node LVTs, and send back results, like a GVT. Transparency of coprocessor parallel services is given by *asynchronous channels* with nodes. These channels are implemented with double-register directional mechanisms connecting the coprocessor to a node. The coprocessor periodically reads or writes channels while node processors execute less intensive write or read operations respectively.

An asynchronous channel from coprocessor to node works as follows. The coprocessor is always allowed to write its own register, and the node to read its own register. Data are transferred from a coprocessor register to a node register when the node does not execute a read operation. In our example (see figure 2), the interface consists of two asynchronous channels:

- **LVT**: from the node to the coprocessor;
- **GVT**: from the coprocessor to the node.

To obtain fast computation cycles, the coprocessor has a pipeline topology in which one stage is associated to one socket. Partial results between stages are embodied in so-called *tokens*. Token communications are asynchronous. After completion of its task, a socket writes a modified token to its righthard neighbor.

The coprocessor pipeline architecture is composed of two parts:

- A *large operative unit* distributed across every socket. This unit executes a systolic computation on an array of values from asynchronous channels. Results are fed back to the pipeline head.
- A *control unit* implemented at the pipeline head. This controller is in charge of the initialization of the operative unit, and the token generation. It also receives computation results from the operative part.

In practical implementation, the control unit and the first operative unit stage can be merged on the same socket. Furthermore, the control unit automaton can drive several operative units.

2.1.2 Coprocessor Service

The coprocessor service is defined by two successive global operations:

$$\begin{array}{lll} NewGVT & := & min_{i=1..n}(LVT_i), \quad \text{where } LVT_i \text{ is defined to be LVT of node } i. \\ GVT_i & := & NewGVT \ for \ i = 1..n \quad \text{where } GVT_i \text{ is defined to be GVT of node } i. \end{array}$$

Each operation is implemented by an operative unit. The first one completes the systolic computation of the minimum by passing the $NewGVT$ result to the control unit. The second operative unit broadcasts the $NewGVT$ value to every node.
Partial results of the two operative units are embodied in a single token:

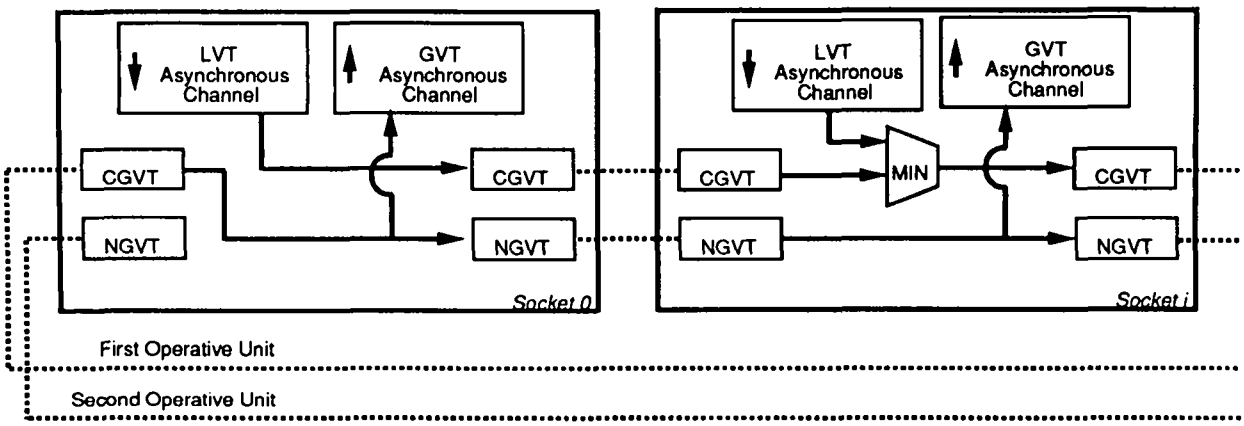

Figure 2: Socket Internal Configuration

Computed Global Virtual Time (CGVT) : partial $NewGVT$ value.
New Global Virtual Time (NGVT) : broadcasted $NewGVT$ value.

The control unit and the first operative unit stages are merged, as shown in *socket 0* of figure 2. It can be seen that the first operative unit is initialized with the LVT value of its socket asynchronous channel at each pipeline cycle. The control unit also feeds the input of the broadcast unit with the output of the minimum computation.

This figure also shows, in the block at socket i , the parallelism between the local minimum and broadcast operations which overlap with one pipeline latency.

3 A hardware service for Time Warp Simulation

This section gives a brief description of *Time Warp* principles as defined in [4] and shows the interest of knowing *Global Virtual Time*.

In the *Time Warp*, all processes are independent and there is no constraint on their asynchronous evolution. Each message is timestamped with the addressed simulation time, t_r. If the LVT of the receiver is already higher than t_r when reception occurs, then the process must roll back to time t_r, and must undo actions between t_r and LVT. All the messages it has sent must be unsent using *anti-messages*. The roll-back mechanism imposes that processes retain the history of states and lists of all messages sent and received. Maintaining all previous information obviously requires an unbounded amount of memory. It has been proved that there is a lower bound on virtual time which the system will never roll-back to [16]. Knowing this lower bound, it is possible to forget older information. This time is called *Global Virtual Time* and is defined as $GVT = min(LVT_i, t_{ri})$, where t_{ri} is the timestamp of message not yet received. GVT must be computed regularly and generally freezes the simulation progress for *one network diffusion time at least*[16]. Notice that LVT has not the same signification as in the previous section: here, LVT is the minimum of all timestamps of one node.

3.1 Algorithm presentation

To implement the Time Warp Simulation, the nodes need to know the Global Virtual Time. Our previous example presents two restrictions: first, there is no provision for a roll back mechanism; secondly, it does not consider time counter overflows. In this section we present a practical solution taking into account these restrictions.

Instead of computing the GVT, this approach tracks a condition where all nodes have overtaken an LVT bound. When this condition is verified, an approximation of GVT has occurred, and then memory garbage collection is possible. To minimize simulation process freezing on memory saturation, the application must tune the GVT progress intervals to deal with node memory capacities and application characteristics. For the sake of simplicity, the intervals between bounds are equal and the same for all processors.

The configurable coprocessor will compute this condition of a global bound overtake.

3.2 Node Message Passing

On message reception, an advanced process can roll back its LVT to a time less than the next bound to overtake. The computation of the condition must take care of unreceived messages. This problem is solved by message acknowledgment.

Each node is supposed to have its current simulation time, and two queues for input and output messages. The current LVT is deduced from the minimum of all time stamps on the node including messages in the input and output queues [4]. Message deletion from an output queue requires an acknowledgment from the communication service to ensure the visibility of the minimum LVT on the coprocessor.Thus, if the bound is not overtaken, it guarantees that there is at least one node which discards this state.

Message passing from node A to B must respect the following protocol:

Node A sends a message from its output queue;
Node B receives this message then places it in its input queue;
Node B computes its new LVT;
Node B sends an acknowledgement to Node A;
Node A receives the acknowledgement;
Node A deletes the message from its output queue;
Node A computes its new LVT.

3.3 Global Condition Computation

A global computation is an operation on an array of values distributed on every socket. This operation can not be instantaneous because of propagation delays. In the previous proposition, local virtual clocks are strictly increasing. So, GVT is evaluated in a systolic way.

In Time Warp Simulation, the condition of a global bound overtake can be expressed as follows. Let us define GO as the Global Overtake condition, and NB the next bound to overtake.

$$GO = min(LVT_{i\ i=1..N}) > NB$$

This expression could be calculated in a systolic way by a distributed operative unit, where PO is defined as a Partial Overtake :

$$PO_0 = true;$$
$$PO_i = PO_{i-1}\ and\ (LVT_i > NB);$$
$$GO = PO_N;$$

Note that the comparison $(LVT_i > NB)$ can be carried out by the node. Only the boolean result is discarded to the socket through a flag O (*Overtake*). So, the operative unit computes the boolean product of the flags O in a systolic way.

With the roll back effect, local virtual clocks are not strictly increasing. A simple systolic computation can provide an erroneous result. The following sequence on message passing from node i to node j illustrates this problem :

Initial conditions : $i > j$; $LVT_j > NB$; $LVT_i < NB$;

1. Computation of PO_j;
2. Node j receives a message from node i provoking a roll back, and decreases LVT_j such $LVT_j < NB$;
3. Node i, receiving the acknowledgement from node j, updates LVT_i such $LVT_i > NB$;
4. Computation of PO_i.

In this example, the operative unit delivers a true GO value to the control unit, but $LVT_j < NB$. Note that at least one of the N following GO will discard a false value.

Property : If N consecutive true GO values are received, the condition of a global bound overtake is true.

Proof: A proof by contradiction can be given. Suppose $W(c)$ is true and $\exists\ O_i(c)$ false with i in $1..N$.
Note $O_i(c)$ the boolean value of flag O of node i at pipeline cycle c.
Then :

$$GO(c) = \bigwedge_{i=1}^{i=N} O_i(c - N + i - 1)$$

And $W(c)$ which is equal to

$$\bigwedge_{\gamma=1}^{\gamma=N} GO(c - \gamma)$$

can be rewritten as :

$$W(c) = \bigwedge_{\gamma=1}^{\gamma=N} \bigwedge_{i=1}^{i=N} O_i(c - \gamma - N + i - 1)$$

so

$$\neg O_i(c) \Rightarrow \exists \neg O_j(c_\delta) \mid c_\delta \in]c - N, c]$$

which is in contradiction with $W(c)$.

To detect this condition two implementations are proposed : when detecting a first true GO, the control unit can either push a marker CO (*Confirm Overtake*) into the pipeline through an operative unit and wait for its return, or count the pipeline cycles to ensure a total dump. The first solution is chosen for the sake of simplicity.

Systolic arrays cannot exactly implement a computation over the array of operands because of the technological depth limitation of reconfigurable sockets. Each token does not operate on simultaneous sample when circulating in the operator. Therefore it is necessary to observe full pipeline results to get accurate conclusions about what has occurred one pipeline delay before.

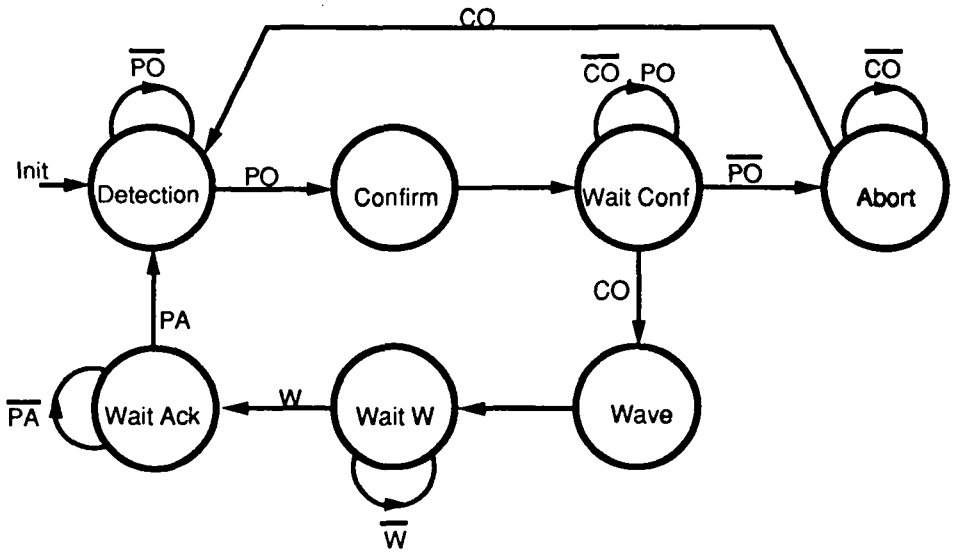

Figure 3: Node 0 Automaton

3.4 Coprocessor Behavior

The general behavior of the coprocessor can be described in three stages:

1. Tracking N consecutive true GO values;
2. Broadcasting the condition of the bound overtake to every node;
3. Waiting for a global acknowledgement from every node.

To implement the second stage, the control unit can push a marker W (*Wave*) through an operative unit and wait for it to come back.

Assuming that each node acknowledges the coprocessor through a flag A, the third stage can be implemented with an operative unit which calculates the boolean product of all the flags in a systolic way. Note that this computation is carried out in a single systolic pass.

In conclusion, the coprocessor consists of four operative units:

PO : computes the boolean product of flags O in a systolic way.
CO : pushes the marker CO through the pipeline.
W : broadcasts the condition of the global bound overtake.
PA : computes the boolean product of flags A in a systolic way.

and a control unit feeding the pipeline with tokens. Figure 3 illustrates the control unit automaton. The transition conditions are flag values from input tokens.

The token structure is:

PO : the partial boolean product of previous node flags O.
CO : a boolean marker to indicate the *Confirm* condition.
W : a boolean marker to indicate the *Wave* condition.
PA : the partial boolean product of previous node flags A.

The control unit can be implemented in the socket 0. The values for the output token of the control

unit are deduced from the automata state:

Detection	: (PO = node.O, CO = false,	W = false,	PA = node.A)
Confirm	: (PO = node.O, CO = true,	W = false,	PA = node.A)
Wait Conf	: (PO = node.O, CO = false,	W = false,	PA = node.A)
Abort	: (PO = node.O, CO = false,	W = false,	PA = node.A)
Wave	: (PO = node.O, CO = false,	W = true,	PA = node.A)
Wait W	: (PO = node.O, CO = false,	W = false,	PA = node.A)
Wait Ack	: (PO = node.O, CO = false,	W = false,	PA = node.A)

The operative units are distributed on each socket executing the following operations on the tokens:

tokenOut.PO = tokenIn.PO *and* flag O
tokenOut.CO = tokenIn.CO
tokenOut.W = tokenIn.W
tokenOut.PA = tokenIn.PA *and* flag A
if tokenIn.W *then* reset flag A

This second example illustrates the use of the coprocessor to compute global conditions in a systolic way by a simple pass, or a pipeline dump. Moreover, the coprocessor controls and sequences actions over the whole network, like broadcasting a condition to every node. Implementation obviously requires very few logic resources, giving way to additive functionalities.

4 Further work : the Armen project

An implementation of the reconfigurable logic layer parallel architecture is currently being built by the LIBr[2]. An MIMD experimental machine called *ArMen* has been designed to investigate most of the capabilities of the architecture. An INMOS T800 has been chosen as the processor node and a Xilinx 3090 LCA [18] as the reconfigurable socket. This leads to small and affordable modules where the socket can operate on addresses, instructions and data from the 32-bit processor multiplexed bus. This is not the most powerful design one could create today, but it is sufficient as a test vehicle. On the other hand, no commercial processor exactly matches our requirements, and there remains a real problem in that we cannot experiment on processor to processor exchanges with the first machine.

Another goal of the ArMen project is to build a software environment for the architecture. Applications can be either specific or general. In the first class, signal processing with use of generic tools, fixed global services like virtual clock support and cellular automaton are considered. Support for these applications can be currently designed as parallel programs and configuration file libraries. The second class of applications is a challenge involving the production of High Level Language development tools for coprocessor synthesis. Given a coprocessor model involving a *node 0* automaton, regular pipelined operative parts and standard interfaces into the node, it is expected that coprocessor generations could be considerably facilitated.

We have shown that the proposed architecture with its accelerated network layer is able to compute global information all over the system with low time cost[3]. It is of obvious interest for the efficient implementation of many applications requiring multiprocessor computation like large logic simulations, signal or image processing, etc...

[2]Laboratoire d'Informatique de Brest is a common structure to *Université de Bretagne Occidentale* and *Ecole Nationale Supérieure des Télécommunications de Bretagne*

[3]pipeline delays are in the order of 50ns

More generally, one can take advantage of the active communication layer to implement every algorithm requiring global knowledge computation. This can be done very simply on the hypothesis of always empty communication channels (of the application layer). Another use of the active layer can be found in the out-of-band communication between sites. Urgent messages can be routed via this layer by a token containing data and destination. The token can be used either for point-to-point communication, partial diffusion (with an associated list) or complete diffusion.

These points are being currently actually studied in the framework of distributed discrete event simulators.

5 Conclusion

The proposed architecture complements current parallel designs on many levels.

A first important property of configurable logic is its ability to synthesize small data-flow sequenced operators, and thus to increase the level of parallelism within the nodes.

The connection of adjacent logic arrays provides a global programmable logic resource, on which very large operative parts with arrays of input/output ports are implemented. These ports handle the whole state of the machine repeatedly by feeding systolic arrays with it. We have shown some internal points of the coprocessors, with *global operators* which are small automata controlling systolic linear parts, as well as asynchronous channels and interrupt waves to interfere with node behavior. These tools are useful in computing global resources, or controlling the whole network behavior.

Accessing global conditions over a distributed system has often been considered to require heavy local computation and communication or synchronization tasks. Pure distributed implementations can fail[6] because of the inefficiency of these mechanisms: communications and local computations are involved in calculating results which must be dispatched back to the nodes. It is expected that the logic layer architecture will encourage the use of efficient global services within MIMD machines for distributed systems, languages or algorithms.

References

[1] P.Bertin, D.Roncin, J.Vuillemin, "Introduction to programmable active memories", in *Systolic Array Processors*, Prentice Hall, pp. 301, 1989.

[2] C.Buzzell, M.J.Robb, R.Fujimoto, "Modular VME rollback approach for Time Warp", in *Proc. of the SCS multiconference on Distributed Simulation*, San Diego, pp. 153, 1990.

[3] A.Gottlieb and al., "The NYU Ultracomputer — Designing an MIMD shared memory parallel computer", in *Proc. International Conference on Computer Architecture*, ACM, pp. 175. 1982.

[4] D.Jefferson, H.Sowizral, "Fast concurrent simulation using the time warp mechanism", in *Proc. of the SCS Conference on Distributed Simulation*, San Diego, pp. 63-69, Jan. 1985.

[5] D.Jefferson, "Virtual Time", *Transactions on Programming, Languages and Systems*, ACM, vol. 7, no. 3, pp. 404, 1985.

[6] R.Fujimoto, J-J.Tsai, G.Gopalakrishnan, "Design and performance of special purpose hardware for Time Warp", in *Proc. of International Symposium on Computer Architecture*, IEEE, pp. 401, 1988.

[7] T.Kean, J.Gray, "Configurable hardware: two case studies of micro-grain calaculation", in *Systolic Array Processors*, Prentice Hall, pp. 310, 1989.

[8] B.W. Lampson, K.A.Pier, "A Processor for a High-Performance Personal Computer", in *Proc. of Computer Architecture Symposium*, IEEE-ACM, pp. 146, 1980.

[9] H.T.Kung, "Why systolic architectures?", *IEEE Computer*, vol. 15, no. 1, pp. 37, 1982.

[10] H.T.Kung, "Network-based multicomputers : redefining high performance computing in the 1990s", in *Proc. of the Decennial Caltech Conference on VLSI*, The MIT Press, pp. 49, 1989.

[11] L.Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System", *Communications of the ACM*, vol. 21, no. 7, pp.558, 1978.

[12] J.Misra, "Distributed Discrete Event Simulation", *Computing Surveys*, vol. 18, no. 1, pp. 39, March 1986.

[13] B.Pottier, D.Lavenier, "High rate sigma filtering, feasibility studies on processor networks", in *Proc. of IFIP Workshop "Parallel architectures on Silicon"*, INP Grenoble, pp. 182, 1989.

[14] B.Pottier, "Machines parallèles à accélérateurs reconfigurables", *Thèse de l'Université de Rennes 1*, Dec. 1990.

[15] M. Raynal, A distributed algorithm to prevent mutual drift between N logical clocks, *Information Processing Letters*, vol. 24, no. 3, pp. 199-202, Feb 1987.

[16] B. Samadi, "Distributed Simulation, Algorithms and Performance Analysis", PhD. Num 8513157, University of California, Los Angeles, pp. 35-64, 1985.

[17] J.Viitanen, T.Kean, "Image pattern recognition using configurable Logic Cell Arrays", in *Proc. of Computer Graphic International '89*, Springer-Verlag, pp. 355, 1989.

[18] Xilinx, The Programmable Gate Array Data Book, *Xilinx*, San Jose, 1990.

LISTE DES DERNIERES PUBLICATIONS INTERNES IRISA

PI 556 CONCEPTION ET INTEGRATION D'UN CORRELATEUR SYSTOLIQUE
Catherine DEZAN, Eric GAUTRIN, Patrice QUINTON
Novembre 1990, 16 Pages.

PI 557 VARIATIONAL APPROACH OF A MAGNETIC SHAPING PROBLEM
Michel CROUZEIX
Novembre 1990, 14 Pages.

PI 558 THE DAVIDSON METHOD
Michel CROUZEIX, Bernard PHILIPPE et Miloud SADKANE
Novembre 1990, 22 Pages.

PI 559 A DISTRIBUTED SOLUTION TO THE k-OUT OF-M RESOURCES ALLOCATION PROBLEM
Michel RAYNAL
Novembre 1990, 18 Pages.

PI 560 A SIMPLE TAXONOMY FOR DISTRIBUTED MUTUAL EXCLUSION ALGORITHMS
Michel RAYNAL
Novembre 1990.

PI 561 MULTIMODAL ESTIMATION OF DISCONTINOUS OPTICAL FLOW USING MARKOV RANDOM FIELDS
Fabrice HEITZ, Patrick BOUTHEMY
Novembre 1990, 50 Pages.

PI 562 EFFICIENT GLOBALCOMPUTATIONS ON A PROCESSOR NETWORK WITH PROGRAMMABLE LOGIC
J M. FILLOQUE, E. GAUTRIN, B POTTIER
Novembre 1990, 14 pages.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique.

ISSN 0249-6399